

Invoking SageMaker machine learning (ML) models from Glue jobs

Ivan Chen, Senior Solutions Architect

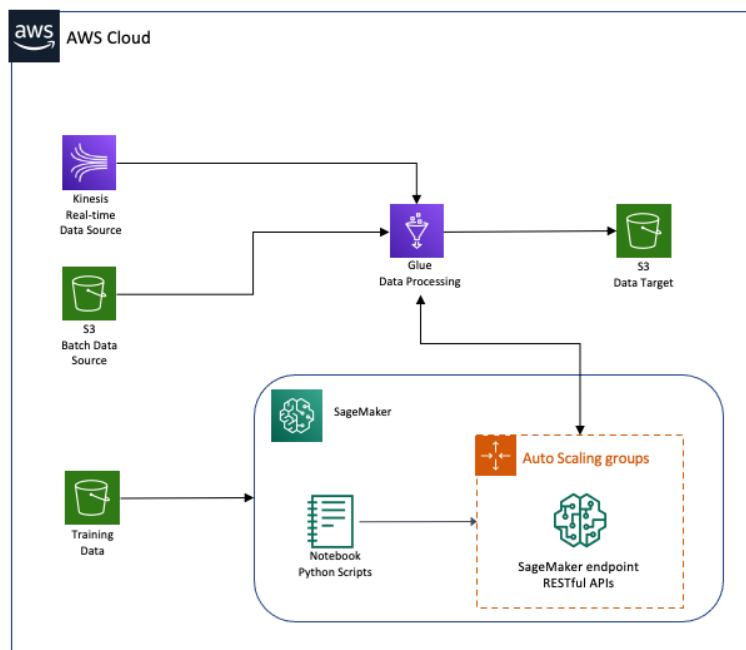
Machine Learning (ML) is getting more and more popular. It helps or enables a lot of organizations to improve their business operations, enable new products or services, or have better decision making. Many organizations have the following problem: Enable machine learning models to be a part of their data processing processes so that they can have a ML-powered analytics pipeline. Usually, the data volume for this type of tasks is very large, such as several-million records (hundreds of GBs or several TBs) per day. How to effectively and efficiently handle such large amount of data with machine learning models is one of challenges that organizations have to face.

With AWS, a lot of organizations are trying to leverage ML model creation on Amazon SageMaker to improve their ML development processes. The AWS Glue service is a popular serverless tool for data engineers and developers to process a large amount of data, which usually is scaled from GB size to TB size daily. How to invoke the ML models hosting on SageMaker from a Glue job for batch and (real-time) streaming processes is a challenge problem. We have helped a lot of developers and data scientists to resolve this challenging problem. In this post, we demonstrate a solution of how to leverage Glue and SageMaker services together to build a ML-powered analytics pipeline.

Overview of solution

The rest sections of this post show readers how to host a machine learning model on a SageMaker endpoint, then create batch and real-time streaming Glue jobs to invoke the machine learning model to generate prediction data along with the original input data, and finally save all the data into a S3 bucket.

The following diagram illustrates the architecture of the solution.



Walkthrough

The following sections will walk through the steps about AWS services setup. This requires some basic machine learning (ML) knowledge and the knowledge on AWS services, such as SageMaker, IAM, Glue and Kinesis. The high-level steps include:

- Create proper IAM roles.
- Create and deploy a Machine Learning (ML) model.
- Create a Glue Spark batch job to invoke the ML model for predictions
- Create a Glue (real-time) streaming job to invoke the ML model for real-time predictions
- Clean up

All the codes included in this post can be found at this [Github repo](#).

Prerequisites

For this walkthrough, you should have the following prerequisites:

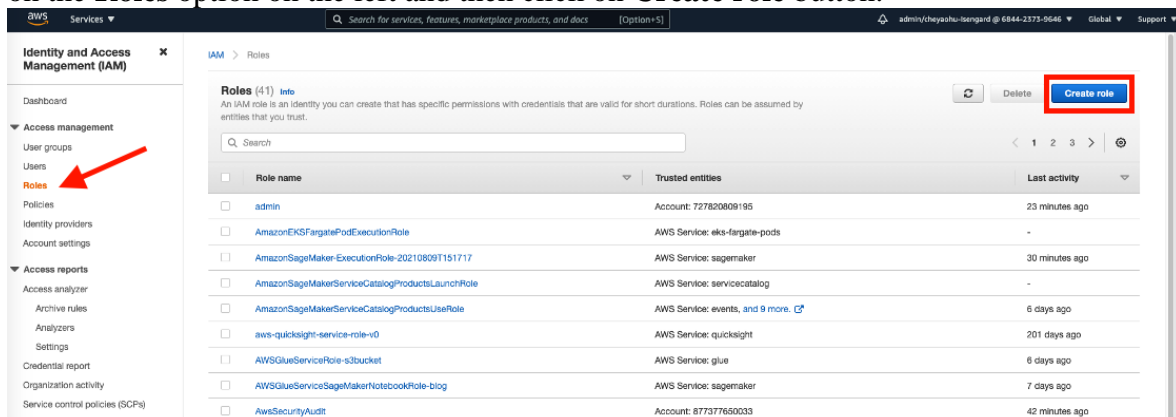
- An [AWS account](#) and the user setup with **AdministratorAccess** permission. For how to create an admin IAM user, please refer to this [AWS on-line document](#).
- AWS resources of [Glue](#), [Kinesis](#) and [SageMaker](#) services
- Basic knowledge on running Python codes on a Jupyter notebook
- Basic knowledge on setting IAM users and roles

Create step section

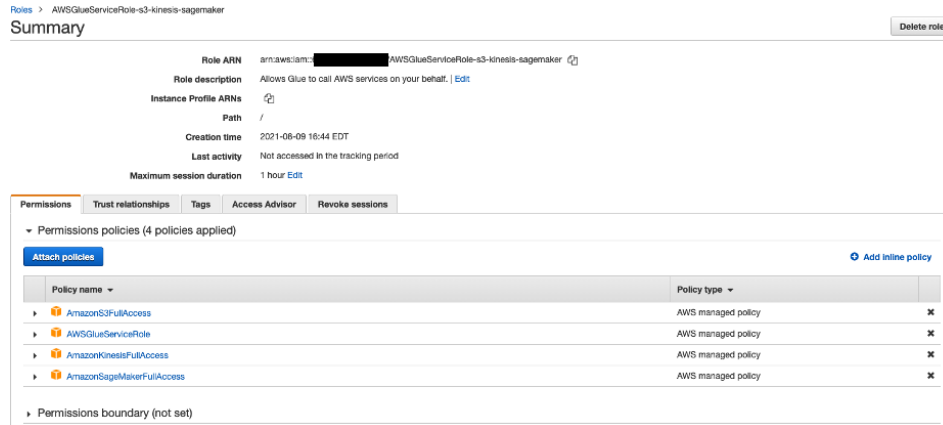
I. Create an IAM role

First of all, you need to setup an IAM role with proper permission policies to allow Glue service to communicate with Kinesis and SageMaker services.

1. Open the [AWS IAM console](#).
2. Click on the **Roles** option on the left and then click on **Create role** button.



3. Choose **AWS service** for trusted entity type and then choose **Glue** for use cases. Click **Next: Permissions** button.
4. Search and then choose the following policies: **AWSGlueServiceRole**, **AmazonKinesisFullAccess**, **AmazonSageMakerFullAccess**. Click **Next: Tags** button. Then click **Next: Review** button.
5. Give the **Role name** as **AWSGlueServiceRole-s3-kinesis-sagemaker** and then click **Create role** button. After this step, you should see the role setup like the screen shot below:



II. Create a Machine Learning Model

You will use SageMaker Studio to build a machine learning model and deploy it on an endpoint. The on-line AWS documentation for onboarding SageMaker studio using Quick Start can be found [here](#).

To onboard to Studio using Quick start

1. Open the [SageMaker console](#).
2. Choose **US East (N. Virginia) us-east-1** region.
3. Choose **Amazon SageMaker Studio** at the top left of the page.
4. On the **SageMaker Studio** page, under **Get started**, choose **Quick start**.
5. For **User name**, type **default-glue-sagemaker**.
6. For **Execution role**, choose **Create a new role**, the **Create an IAM role** dialog opens, like the screen shot below:

Create an IAM role

Passing an IAM role gives Amazon SageMaker permission to perform actions in other AWS services on your behalf. Creating a role here will grant permissions described by the [AmazonSageMakerFullAccess](#) IAM policy to the role you create.

The IAM role you create will provide access to:

- ☒ **S3 buckets you specify - optional**
 - ☒ **Any S3 bucket**
Allow users that have access to your notebook instance access to any bucket and its contents in your account.
 - ☐ **Specific S3 buckets**

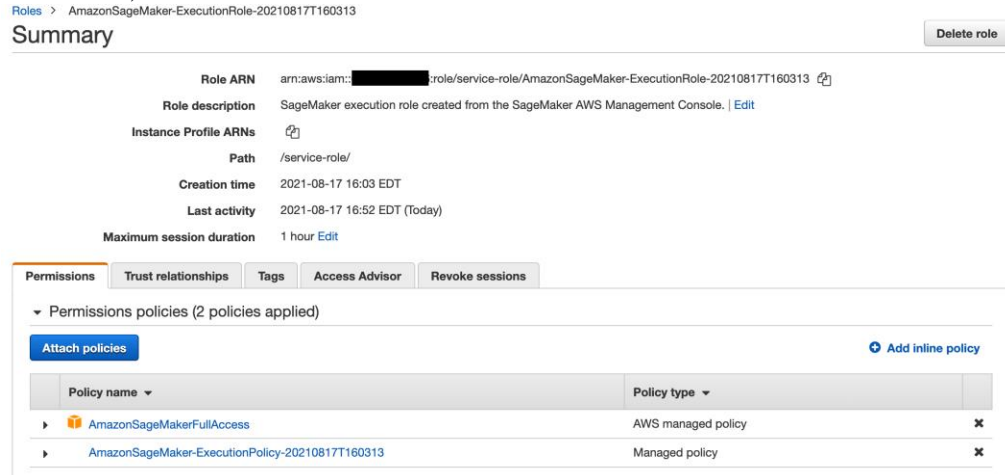
Example: `bucket-name-1, bucket-name-2`

Comma delimited. ARNs, "*" and "/" are not supported.
 - ☐ **None**
- ☒ **Any S3 bucket with "sagemaker" in the name**
- ☒ **Any S3 object with "sagemaker" in the name**
- ☒ **Any S3 object with the tag "sagemaker" and value "true"**
[See Object tagging](#)
- ☒ **S3 bucket with a Bucket Policy allowing access to SageMaker**
[See S3 bucket policies](#)

Cancel

Create role

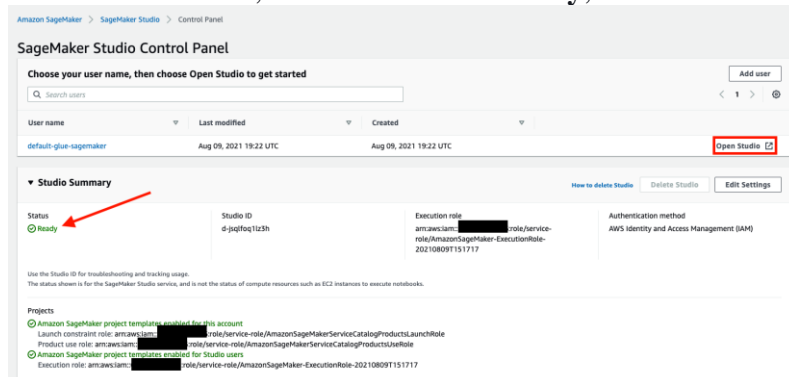
- Keep default option on **Any S3 bucket**.
- Click **Create role** button. SageMaker will create a new IAM AmazonSageMaker-ExecutionPolicy role with the **AmazonSageMakerFullAccess** and **AmazonKinesisFullAccess** policies attached, which is similar to the screen shot below.



7. Click **Submit** button.



Note: If you receive an error message that you need to create a VPC, see [Choose a VPC](#).

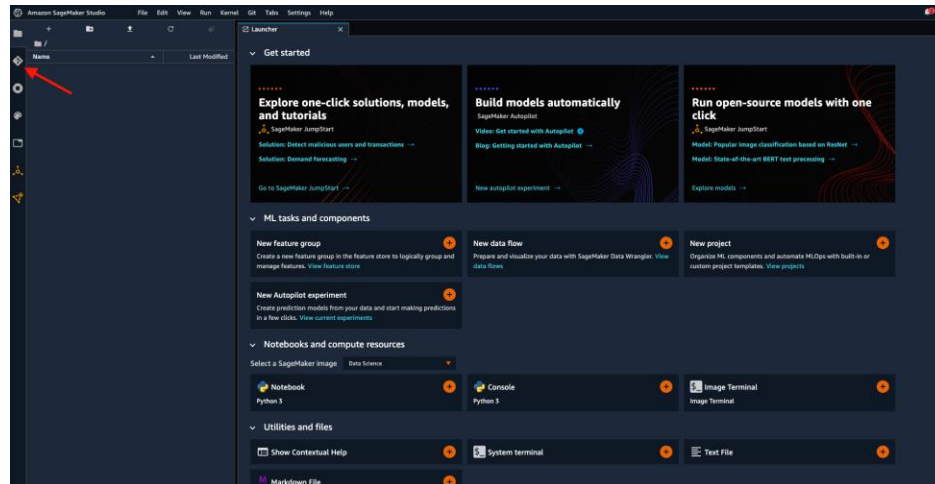
On the SageMaker Studio Control Panel, under **Studio Summary**, wait for Status to change to **Ready**.



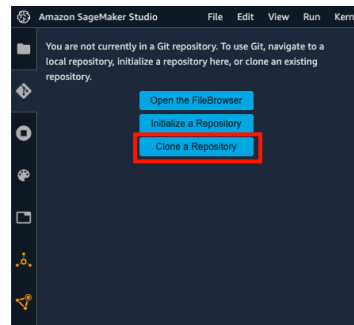
When Status is Ready, the username **default-glue-sagemaker** is enabled and chosen. Choose **Open Studio**. The Amazon SageMaker Studio loading page displays.

To clone a repo to create & deploy a ML model on a Sagemaker endpoint

1. In the left sidebar of SageMaker Studio webpage (shown above), choose the **File Browser** icon ().
2. Choose the root folder or the folder you want to clone the repo into.
3. In the left sidebar, choose the **Git** icon ().



4. Choose **Clone a Repository**.



5. Enter the URI for the Github repo <https://github.com/chen115y/Glue-sagemaker.git>.
6. Choose **CLONE**.
7. Wait for the download to finish. After the repo has been cloned, the **File Browser** opens to display the cloned repo.
8. Double click the jupyter notebook **xgboost_customer_churn.ipynb** and run all cells on it.
9. If all cells run successfully without any errors, you should see a SageMaker endpoint **sagemaker-glue-blog-xgboost-churn** is in the **InService** status like following screen shot.



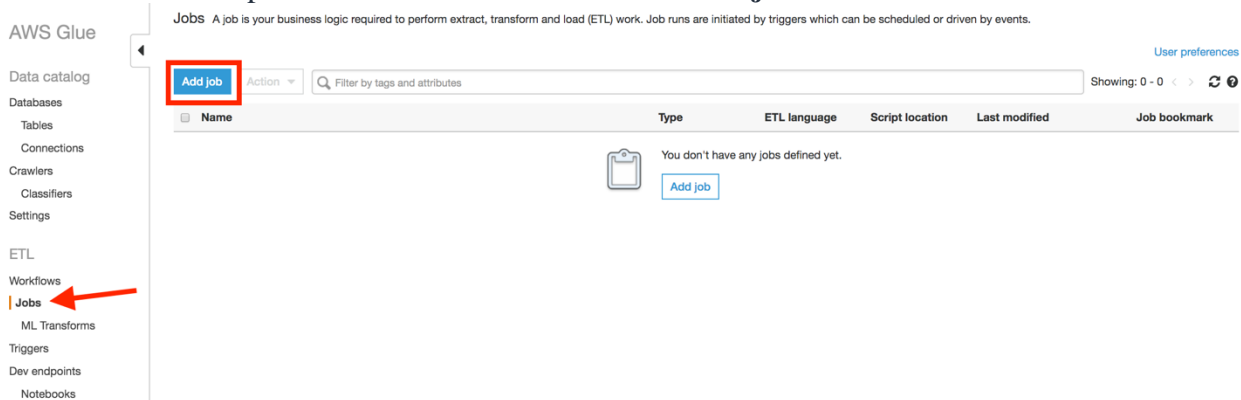
Here the jupyter notebook describe using machine learning (ML) for the automated identification of unhappy customers, also known as customer churn prediction. It contains the codes for training data preparation, model training and model deployment on SageMaker. The ML algorithm used in this jupyter notebook is XGBoost.

Note: after the jupyter notebook has been executed, the churn.txt data file has been uploaded to the S3 bucket, which follows the pattern of “s3://sagemaker-us-east-1-<<AWS account number>>/sagemaker/DEMO-xgboost-churn/churndata/churn.txt”. The AWS account number in this pattern is a 12-digit number account Id and can be found at the [Account Settings](#) page.

III. Create a Glue Spark Batch Job to Invoke the ML Model

In this section, you will create a Spark job in Glue, which will read input data from a S3 bucket, invoke the SageMaker ML model endpoint to generate the predictions, combine the original input data and the predictions together and then write them to a S3 bucket in csv format.

1. Open the [AWS Glue console](#).
2. Make sure you choose **US East (N. Virginia) us-east-1** region.
3. Click on the Jobs option on the left and then click on **Add job** button.



4. Give the name for your job as **glue-sagemaker-batch-job**,
5. For IAM role, choose **AWSGlueServiceRole-s3-kinesis-sagemaker** from the dropdown list.
6. For This job runs, choose **A new script to be authored by you**. Like the screen shot below. And then click **Next** button.

The screenshot shows the 'Configure the job properties' form in the AWS Glue console. The form fields are as follows:

- Name:** glue-sagemaker-spark-job
- IAM role:** AWSGlueServiceRole-s3-kinesis-sagemaker
- Type:** Spark
- Glue version:** Spark 2.4, Python 3 with improved job startup times (Glue Version 2.0)
- This job runs:** A new script to be authored by you (selected)
- Script file name:** glue-sagemaker-spark-job
- S3 path where the script is stored:** s3://aws-glue-scripts/[redacted]-us-east-1/admin
- Temporary directory:** s3://aws-glue-temporary/[redacted]-us-east-1/admin

Below these fields are expandable sections for 'Advanced properties', 'Monitoring options', 'Tags (optional)', 'Security configuration, script libraries, and job parameters (optional)', and 'Catalog options (optional)'. A 'Next' button is at the bottom right.

7. Click **Save job and edit script** button. On the glue script edit screen, copy and paste the following Python script.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from awsglue.job import Job
from awsglue.dynamicframe import DynamicFrame
import boto3
from pyspark.context import SparkContext
from pyspark.sql.functions import col, udf, struct
from pyspark.sql.types import *

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

# get job parameter values
region = 'us-east-1'
s3_target = 's3://sagemaker-us-east-1-<<AWS account number>>/sagemaker/DEMO-xgboost-churn/result/'
s3_source = 's3://sagemaker-us-east-1-<<AWS account number>>/sagemaker/DEMO-xgboost-
churn/churndata/churn.txt'
endpoint_name = 'sagemaker-glue-blog-xgboost-churn'

# read data from source
df = spark.read.option("header", True).csv(s3_source)
df = df.drop('Phone', 'Day Charge', 'Eve Charge', 'Night Charge', 'Intl Charge', 'Churn?') \
    .select('State', 'Account Length', 'Area Code', 'Intl Plan', 'VMail Plan', 'VMail Message', 'Day Mins', 'Day
Calls', 'Eve Mins', 'Eve Calls', 'Night Mins', 'Night Calls', 'Intl Mins', 'Intl Calls', 'CustServ Calls')
# df.printSchema()

def OneHot_Encoding(column_name, tmp):
    categories = tmp.select(column_name).distinct().rdd.flatMap(lambda x : x).collect()
    categories.sort()
    for category in categories:
        function = udf(lambda item: 1 if item == category else 0, IntegerType())
        new_column_name = column_name + '_' + category
        tmp = tmp.withColumn(new_column_name, function(col(column_name)))
    tmp = tmp.drop(column_name)
    return tmp

df = OneHot_Encoding("State", df)
df = OneHot_Encoding("Intl Plan", df)
df = OneHot_Encoding("VMail Plan", df)

def get_prediction(row):
    infer_data = ','.join([str(elem) for elem in list(row)])
    runtime_client = boto3.client('runtime.sagemaker', region)
    response = runtime_client.invoke_endpoint(EndpointName=endpoint_name, ContentType="text/csv",
    Body=infer_data)
    result = response["Body"].read()
```



```

result = result.decode("utf-8")
return int(float(result) > 0.5)

pred_udf = udf(lambda x: get_prediction(x) if x is not None else None, IntegerType())
df_result = df.withColumn("prediction", pred_udf(struct([df[x] for x in df.columns])))

applymapping1 = DynamicFrame.fromDF(df_result, glueContext, 'applymapping1')
datasink2 = glueContext.write_dynamic_frame.from_options(frame = applymapping1, connection_type = "s3",
connection_options = {"path": s3_target}, format = "csv", transformation_ctx = "datasink2")

job.commit()

```

Note: for the codes, you need to replace all occurrences of <<AWS account number>> with your 12-digit AWS account Id, which can be found at the [Account Settings](#) page.

The Glue job codes above will do some one-hot encoding transformation on the data set first. Then it is defining a function called *get_prediction* to invoke the Sagemaker ML endpoint to do the real-time prediction and a [Spark User Defined Function \(UDF\)](#) to leverage the *get_prediction* function on data transformation. The UDF applies on each row of data and adds the prediction result as additional column called “prediction” as codes shown below.

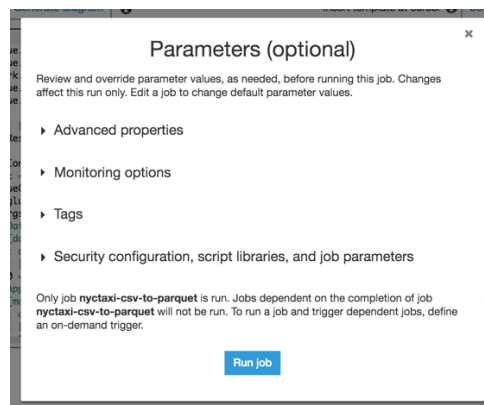
```

def get_prediction(row):
    infer_data = ','.join([str(elem) for elem in list(row)])
    runtime_client = boto3.client('runtime.sagemaker', region)
    response = runtime_client.invoke_endpoint(EndpointName=endpoint_name, ContentType="text/csv", Body=infer_data)
    result = response["Body"].read()
    result = result.decode("utf-8")
    return int(float(result) > 0.5)

pred_udf = udf(lambda x: get_prediction(x) if x is not None else None, IntegerType())
df_result = df.withColumn("prediction", pred_udf(struct([df[x] for x in df.columns])))

```

8. Click **Save** button to save the job and then click **Run job** button. It should open up the window below. Here, you can provide additional configurations for the job. You can pass Tags, update different monitoring options, provide library path or configure worker type and resource size. For this exercise, leave all options as default and click on **Run job** button.



9. When the job is running, you will notice that the Run job button is grayed out with a circular-spinner icon.


```

1 import sys
2 from awslogs.transforms import *
3 from awslogs.utils import getResolvedOptions
4 from awslogs.context import GlueContext
5 from awslogs.job import Job
6 from awslogs.dynamicframe import DynamicFrame
7 import boto3
8 from pyspark.context import SparkContext
9 from pyspark.sql.functions import col, udf, struct
10 from pyspark.sql.types import *
11
12 ## @param: [JOB_NAME]
13 args = getResolvedOptions(sys.argv, ['JOB_NAME'])
14
15 sc = SparkContext()
16 glueContext = GlueContext(sc)
17 spark = glueContext.spark_session
18 job = Job(glueContext)
19 job.init(args['JOB_NAME'], args)
20
21 # get job parameter values
22 region = 'us-east-1'
23 s3_target = 's3://sagemaker-us-east-1-684423739646/sagemaker/DEMO-xgboost-churn/result/'
24 s3_source = 's3://sagemaker-us-east-1-684423739646/sagemaker/DEMO-xgboost-churn/churndata/churn.txt'
25 endpoint_name = 'sagemaker-glue-logs-xgboost-churn'
26
27 # read data from source
28 df = spark.read.option('header', 'True').csv(s3_source)
29 df = df.drop('Phone', 'Day Charge', 'Even Charge', 'Night Charge', 'Intl Charge', 'Churn') \
30     .select('State', 'Account Length', 'Area Code', 'Intl Plan', 'Mail Plan', 'Mail Message', 'Day Mins', 'Day Calls', 'Even Mins', 'Even Calls', 'Night Mins', 'Night Calls', 'Intl Mins', 'Intl Calls', 'CustomServ
31 # df.printSchema()
32
33 - def OneHot_Encoding(column_name, tmp):
34     categories = tmp.select(column_name).distinct().rdd.flatMap(lambda x: x).collect()
35     categories.sort()
36     for category in categories:
37         function = udf(lambda item: 1 if item == category else 0, IntegerType())
38         new_column_name = column_name + "_" + category
39         tmp = tmp.withColumn(new_column_name, function(col(column_name)))
40     tmp = tmp.drop(column_name)
41     return tmp
42
43 df = OneHot_Encoding('State', df)
44 df = OneHot_Encoding('Area Code', df)
45 df = OneHot_Encoding('Intl Plan', df)
46 df = OneHot_Encoding('Mail Plan', df)

```

10. Exit from the script window. You can monitor additional job metrics by going back to the Glue job console and select the individual job.

Jobs A job is your business logic required to perform extract, transform and load (ETL) work. Job runs are initiated by triggers which can be scheduled or driven by events.

[User preferences](#)

[Add job](#) [Action](#) Showing: 1 - 2

<input type="checkbox"/> Name	Type	ETL language	Script location	Last modified	Job bookmark
<input checked="" type="checkbox"/> glue-sagemaker-batch-job	Spark	python	s3://aws-glue-s...	12 August 2021 8:53 PM ...	Disable
<input type="checkbox"/> glue-sagemaker-stream-job	Spark Streaming	python	s3://aws-glue-s...	3 August 2021 12:51 PM ...	Disable

[History](#) [Details](#) [Script](#) [Metrics](#)

[View run metrics](#) [Rewind job bookmark](#) Showing: 1 - 2

Run ID	Retry attempt	Run status	Error	Output	Logs	Error logs	Glue version	Maximum capacity	Triggered by	Start time	End time	Start-up time	Execution time	Timeout	Delay	Job run input
<input type="radio"/> jr_0a882fb14b...	-	Running			Logs	Error logs	2.0	2		12 A...		0 secs	1 min	2880 mins		s3://aws-glue-s...

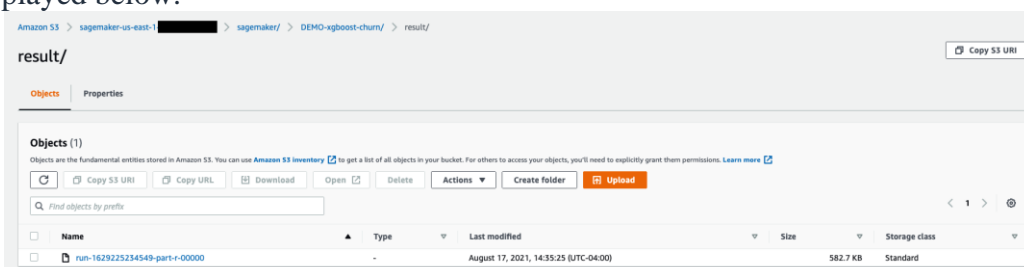
11. **Run status** column shows job status. This Spark job may take about 3 minutes. Once the job is completed successfully, it will show up Succeeded status.

[History](#) [Details](#) [Script](#) [Metrics](#)

[View run metrics](#) [Rewind job bookmark](#) Showing: 1 - 2

Run ID	Retry attempt	Run status	Error	Output	Logs	Error logs	Glue version	Maximum capacity	Triggered by	Start time	End time	Start-up time	Execution time	Timeout	Delay	Job run input
<input type="radio"/> jr_0a882fb14b0c0055...	-	Succeeded			Logs	Error logs	2.0	2		12 Augu...	12 Augu...	7 secs	3 mins	2880 mins		s3://aws-glue-tempor...

12. If the job is finished successfully, it should have created csv output in the target location, which is something like 's3://sagemaker-us-east-1-<AWS account number>/sagemaker/DEMO-xgboost-churn/result/'. Open up the S3 console and check the target location. The location should be populated with a csv file as displayed below.



If you crawl the data file via [Blue Crawler](#) and/or use [Athena](#) to query the data, you should see the prediction column as displayed below.

The screenshot shows the Amazon Athena console interface. At the top, there's a query editor with a SQL query: `1 SELECT * FROM "default"."result" limit 10;`. Below the query, there are buttons for 'Run query', 'Save as', and 'Create'. A status bar indicates '(Run time: 0.64 seconds, Data scanned: 299.93 KB)'. Below the query editor, there's a 'Results' section showing a table with 16 columns. The columns are: `sc`, `state_sd`, `state_tn`, `state_tx`, `state_ut`, `state_va`, `state_vt`, `state_wa`, `state_wi`, `state_wv`, `state_wy`, `int'l plan_no`, `int'l plan_yes`, `vmail plan_no`, `vmail plan_yes`, and `prediction`. The `prediction` column is highlighted with a red box. The table contains 10 rows of data, all with '0' in the `prediction` column.

sc	state_sd	state_tn	state_tx	state_ut	state_va	state_vt	state_wa	state_wi	state_wv	state_wy	int'l plan_no	int'l plan_yes	vmail plan_no	vmail plan_yes	prediction
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0

IV. Glue Spark Streaming Job for Real-time Data Processing

In this section, you will create a Spark streaming job in Glue, which will read input data from a Kinesis data stream, invoke the SageMaker ML model endpoint to generate the predictions, combine the original input data and the predictions together and then write them to a S3 bucket in csv format.

Kinesis Real-time data

First of all, you need to create a Kinesis data stream and then ingest the churn data into it so that the Glue job can read the data from the Kinesis. This can be done by running the notebook **kinesis-put-get-records.ipynb**.

1. In order to run the notebook, you need to grant SageMaker permissions to access Kinesis data stream.

Go to SageMaker Studio, find the **Execution role** on Control Panel as shown below.

The screenshot shows the SageMaker Studio Control Panel. The 'Studio Summary' section is expanded, showing details about the studio. The 'Execution role' is highlighted with a red box. The execution role is `arn:aws:iam::684423739646:role/service-role/AmazonSageMaker-ExecutionRole-20210817T160313`. The 'Authentication method' is 'AWS Identity and Access Management (IAM)'. The 'Status' is 'Ready'. The 'Studio ID' is `d-srosumc3cz9`. Below the studio summary, there's a 'Projects' section showing the 'Amazon SageMaker project templates enabled for this account' and the 'Amazon SageMaker project templates enabled for Studio users'.

SageMaker Studio Control Panel

Choose your user name, then choose Open Studio to get started

Search users

User name	Last modified	Created	
default-glue-sagemaker	Aug 17, 2021 20:06 UTC	Aug 17, 2021 20:06 UTC	Open Studio

Studio Summary

Status: Ready

Studio ID: d-srosumc3cz9

Execution role: arn:aws:iam::684423739646:role/service-role/AmazonSageMaker-ExecutionRole-20210817T160313

Authentication method: AWS Identity and Access Management (IAM)

Use the Studio ID for troubleshooting and tracking usage. The status shown is for the SageMaker Studio service, and is not the status of compute resources such as EC2 instances to execute notebooks.

Projects

Amazon SageMaker project templates enabled for this account

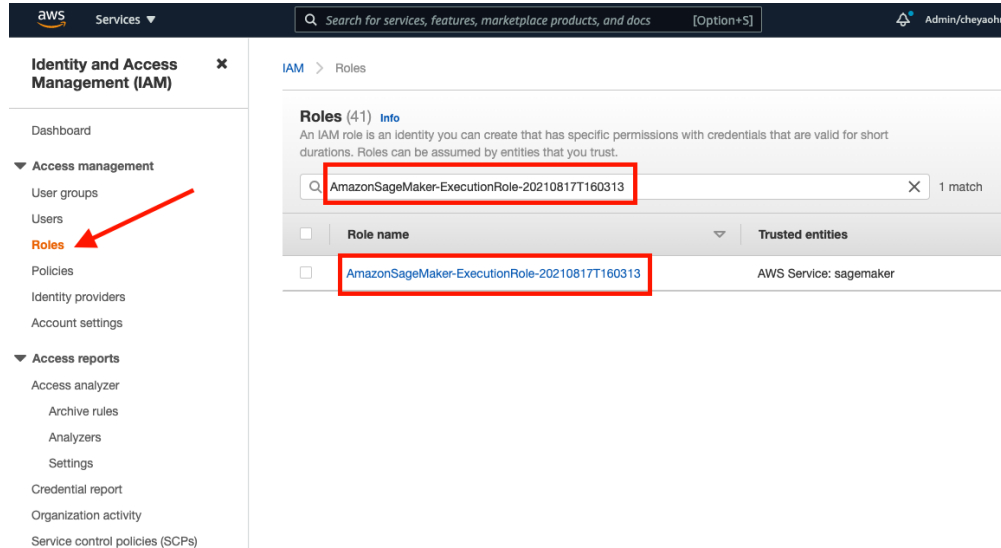
Launch constraint role: arn:aws:iam::684423739646:role/service-role/AmazonSageMakerServiceCatalogProductsLaunchRole

Product use role: arn:aws:iam::684423739646:role/service-role/AmazonSageMakerServiceCatalogProductsUserRole

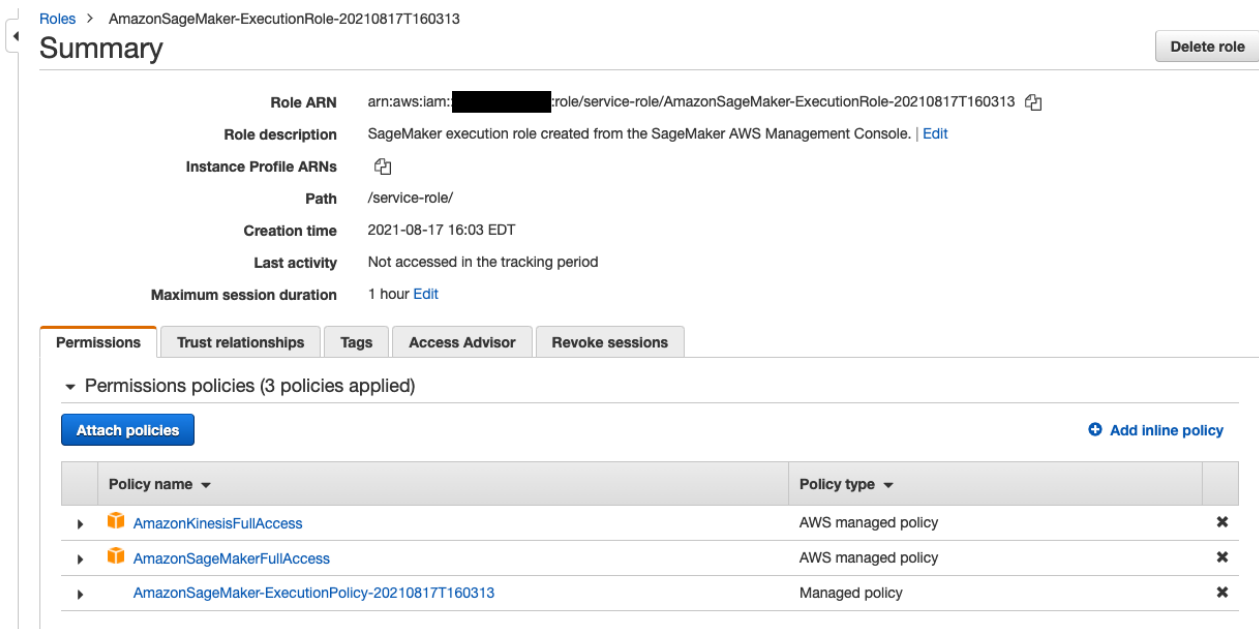
Amazon SageMaker project templates enabled for Studio users

Execution role: arn:aws:iam::684423739646:role/service-role/AmazonSageMaker-ExecutionRole-20210817T160313

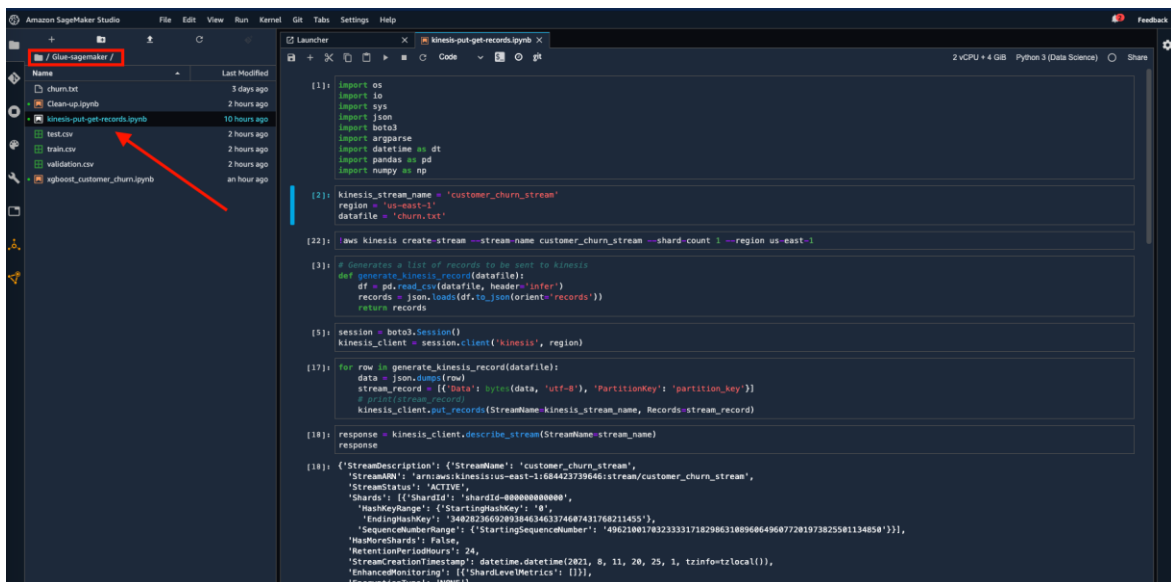
- Open the [AWS IAM console](#).
- Click on the **Roles** option on the left and then search the IAM role that you found on the SageMaker Studio Control Panel.



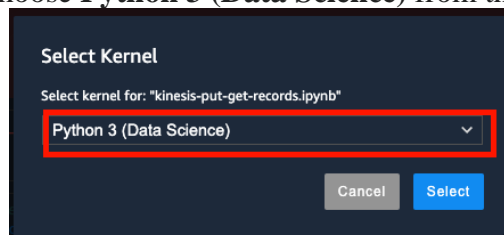
- Click the role and click **Attach policies**.
- Search and then choose the following policy: **AmazonKinesisFullAccess**. Click **Attach policy** button. After this step, you should see the role setup like the screen shot below:



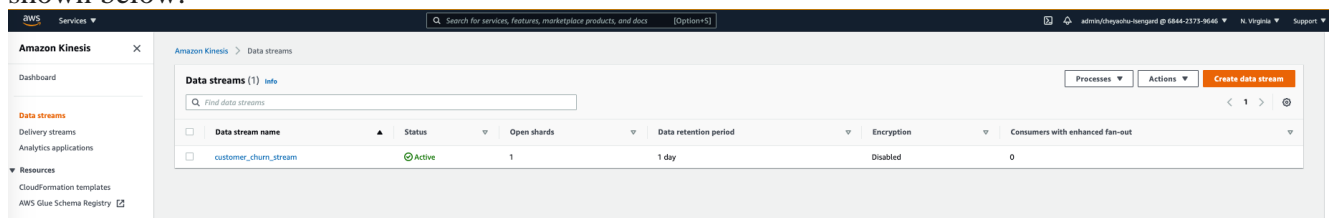
- After you grant the proper permissions on the IAM role, you need to run the notebook to create the Kinesis data stream and ingest the churn data in it: Go back to SageMaker Studio, **Open Studio** for the user **default-glue-sagemaker**.
- Go to the repo folder **Glue-sagemaker**, double click the notebook **kinesis-put-get-records.ipynb** as shown below.



8. For **Select Kernel**, please choose **Python 3 (Data Science)** from the dropdown list.



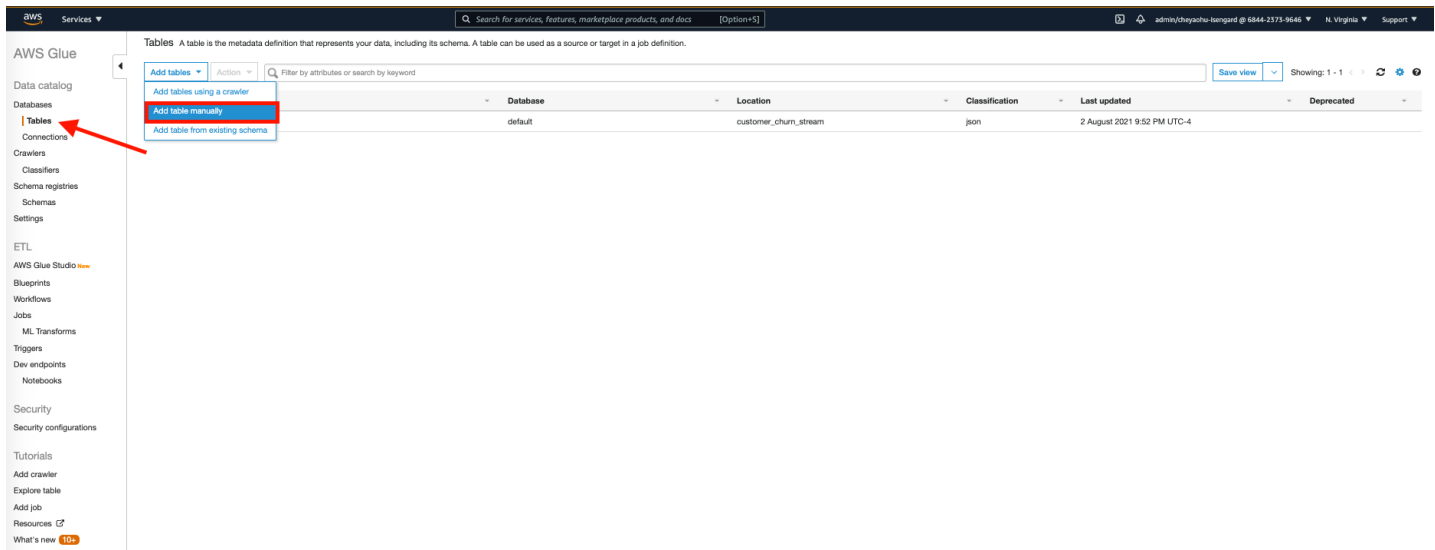
9. Run all cells. If there are no error, the Kinesis data stream has been successfully created and ingested the churn data already. If you go to Kinesis console, you should see an active Kinesis data stream as shown below.



Create data source table on Glue Data Catalog for Kinesis real-time data

In order to let a Glue Spark steam job to get data from Kinesis, you need to set up the data source table on Glue Data Catalog for Kinesis.

1. Open the [AWS Glue console](#).
2. Make sure you choose **US East (N. Virginia) us-east-1** region.
3. Click on the **Tables** option on the left and then click on **Add tables** button to choose **Add table manually**.



- a. For **Table name**, type **kinesis_customer_churn**. Choose or create **default** for **Database** as shown below. And then click **Next** button.

The form titled 'Set up your table's properties' contains the following fields:

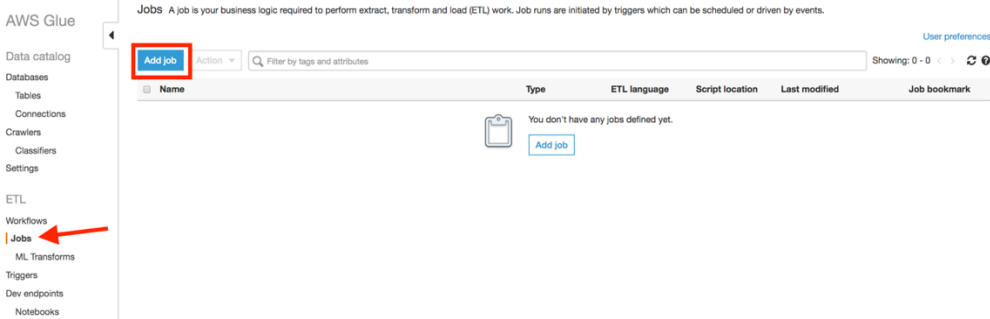
- Table name:** kinesis_customer_churn
- Database:** default
- Buttons:** Add database, Next
- Optional field:** Description (optional)

- b. Choose **Kinesis** on Select the type of source. Choose **Stream in my account** on Select a kinesis data stream. Choose **us-east-1** for region and choose **customer_churn_stream** on Kinesis stream name dropdown list. After that you should see the same webpage as shown below. Then click **Next** button.

The form titled 'Add a data store' contains the following fields:

- Select the type of source:**
 - ☐ S3
 - ☒ Kinesis
 - ☐ Kafka
- Select a kinesis data stream:**
 - ☒ Stream in my account
 - ☐ Stream in another account
- Region:** US East (N. Virginia) us-east-1
- Kinesis stream name:** customer_churn_stream
- Sample size (optional):** Enter an integer between 1 and 249.
- Buttons:** Back, Next

- c. Choose **JSON** for Classification and then click **Next** button.
- d. In the **Define a schema** page, add columns as shown below.



4. Give the name for your job as **glue-sagemaker-streaming-job**,
5. For IAM role, choose **AWSGlueServiceRole-s3-kinesis-sagemaker** from the dropdown list.
6. For (job) Type, choose **Spark Streaming** from the dropdown list.
7. For This job runs, choose **A new script to be authored by you**, like the screen shot below. And then click **Next** button.

Configure the job properties

Name

glue-sagemaker-streaming-job

IAM role

AWSGlueServiceRole-s3-kinesis-sagemaker

Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job. [Create IAM role.](#)

Type

Spark Streaming

Glue version

Spark 2.4, Python 3 with improved job startup times (Glue Version 2.0)

This job runs

☐ A proposed script generated by AWS Glue
 ☐ An existing script that you provide
 ☒ A new script to be authored by you

Script file name

glue-sagemaker-streaming-job

S3 path where the script is stored

s3://aws-glue-scripts-[redacted]-us-east-1/admin

Temporary directory

s3://aws-glue-temporary-[redacted]-us-east-1/admin

▶ Advanced properties
 ▶ Monitoring options
 ▶ Tags (optional)
 ▶ Security configuration, script libraries, and job parameters (optional)
 ▶ Catalog options (optional)

Next

8. Click **Save job and edit script** button. On the glue script edit screen, copy and paste the following Python script.

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql import DataFrame, Row
from pyspark.sql.functions import col, udf, struct
from pyspark.sql.types import *
import datetime
from awsglue import DynamicFrame
import numpy as np
import boto3

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)

## The main process start here ##
endpoint_name = "sagemaker-glue-blog-xgboost-churn"

## Define the One-hot encoding method
def OneHot_Encoding(column_name, tmp):
    categories = tmp.select(column_name).distinct().rdd.flatMap(lambda x : x).collect()
    categories.sort()
    for category in categories:
        function = udf(lambda item: 1 if item == category else 0, IntegerType())
        new_column_name = column_name + '_' + str(category)
        tmp = tmp.withColumn(new_column_name, function(col(column_name)))
    tmp = tmp.drop(column_name)
    return tmp

## Define inference/prediction method via invoking sagemaker endpoint
def get_prediction(row):
    infer_data = ','.join([str(elem) for elem in list(row)])
    sagemaker_client = boto3.client('runtime.sagemaker', 'us-east-1')
    response = sagemaker_client.invoke_endpoint(EndpointName=endpoint_name, ContentType="text/csv",
    Body=infer_data)
    result = response["Body"].read()
    result = result.decode("utf-8")
    return int(float(result) > 0.5)
pred_udf = udf(lambda x: get_prediction(x) if x is not None else None, IntegerType())

## @type: DataSource
## @args: [stream_type = kinesis, stream_batch_time = "100 seconds", database = "default", additionalOptions
= {"startingPosition": "TRIM_HORIZON", "inferSchema": "false"}, stream_checkpoint_location = "s3://sagemaker-
us-east-1-684423739646/sagemaker/DEMO-xgboost-churn/result-stream/checkpoint/", table_name =
"kinesis_customer_churn"]
## @return: datasource0
```

```

## @inputs: []
data_frame_datasource0 = glueContext.create_data_frame.from_catalog(database = "default", table_name =
"kinesis_customer_churn", transformation_ctx = "datasource0", additional_options = {"startingPosition":
"TRIM_HORIZON", "inferSchema": "true"})
def processBatch(data_frame, batchId):
    if (data_frame.count() > 0):
        datasource0 = DynamicFrame.fromDF(data_frame, glueContext, "from_data_frame")
        # ML transformation to prepare the data for inference/prediction
        df = datasource0.toDF()
        df = df.drop('Phone', 'Day Charge', 'Eve Charge', 'Night Charge', 'Intl Charge',
'Churn?').select('State', 'Account Length', 'Area Code', 'Int'l Plan', 'VMail Plan', 'VMail Message', 'Day Mins', 'Day
Calls', 'Eve Mins', 'Eve Calls', 'Night Mins', 'Night Calls', 'Intl Mins', 'Intl Calls', 'CustServ Calls')
        df = OneHot_Encoding("State", df)
        df = OneHot_Encoding("Int'l Plan", df)
        df = OneHot_Encoding("VMail Plan", df)
        # Invoke SageMaker endpoint to do ML prediction
        df_pred = df.withColumn("prediction", pred_udf(struct([df[x] for x in df.columns])))
        datasource1 = DynamicFrame.fromDF(df_pred, glueContext, "datasource1")
        # Save the final results into s3 bucket
        now = datetime.datetime.now()
        year = now.year
        month = now.month
        day = now.day
        hour = now.hour
        minute = now.minute
        path_datasink1 = "s3://sagemaker-us-east-1-

```

Note: for the codes, you need to replace all occurrences of <<AWS account number>> with your 12-digit AWS account Id, which can be found at the [Account Settings](#) page.

- Click the **Save** button to save the job and then click **Run job** button. It should open up the window below. For this exercise, leave all options as default and click on **Run job** button.

✕

Parameters (optional)

Review and override parameter values, as needed, before running this job. Changes affect this run only. Edit a job to change default parameter values.

- ▶ Advanced properties
- ▶ Monitoring options
- ▶ Security configuration, script libraries, and job parameters

Only job **glue-sagemaker-stream-job** is run. Jobs dependent on the completion of job **glue-sagemaker-stream-job** will not be run. To run a job and trigger dependent jobs, define an on-demand trigger.

Run job

10. When the job is running, you will notice that the Run job button is grayed out with a circular-spinner icon.



11. Exit from the script window. You can monitor additional job metrics by going back to the Glue job console and select the individual job.

Add job

Action

Filter by tags and attributes

Showing: 1 - 2

<input type="checkbox"/> Name	Type	ETL language	Script location	Last modified	Job bookmark
<input type="checkbox"/> glue-sagemaker-batch-job	Spark	python	s3://aws-glue-scripts...	12 August 2021 8:53 PM UTC-4	Disable
<input checked="" type="checkbox"/> glue-sagemaker-stream-job	Spark Streaming	python	s3://aws-glue-scripts...	3 August 2021 12:51 PM UTC-4	Disable

History

Details

Script

Metrics

View run metrics

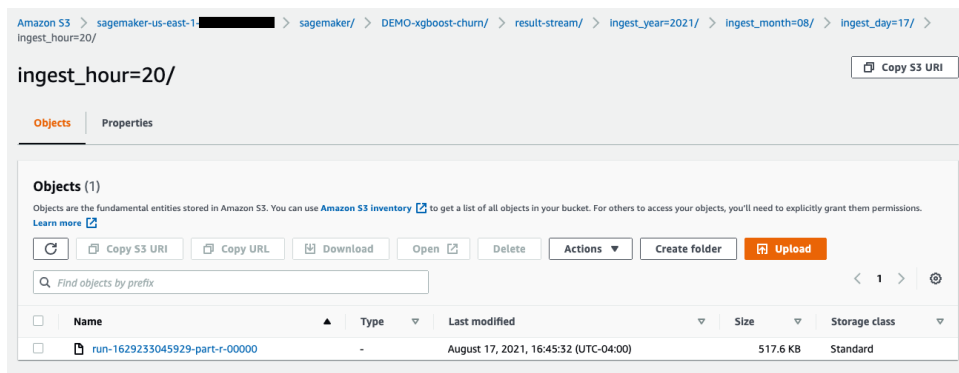
Rewind job bookmark

Showing: 1 - 11

Run ID	Retry attempt	Run status	Error	Output	Logs	Error logs	Glue version	Maximum capacity	Triggered by	Start time	End time	Start-up time	Execution time	Timeout	Delay	Job run input
<input type="radio"/> jr_ebdd244ae5b74f...	-	Running			Logs	Error logs	2.0	4		12 Aug...		0 secs	0 secs			undefined

12. **Run status column** shows job status, this Spark job may take 3 minutes. Then stop the job.

13. If the job is running successfully, it should have created csv output in the target location. Open up the S3 console and check the target location, which follows the pattern of 's3://sagemaker-us-east-1-*<AWS account number>*/sagemaker/DEMO-xgboost-churn/result-stream/ingest_year=*<year>*/ingest_month=*<month>*/ingest_day=*<day>*/ingest_hour=*<hour>*/', like the screen shot displayed below. The data file here should contain the same result data as Glue batch job that you created above.



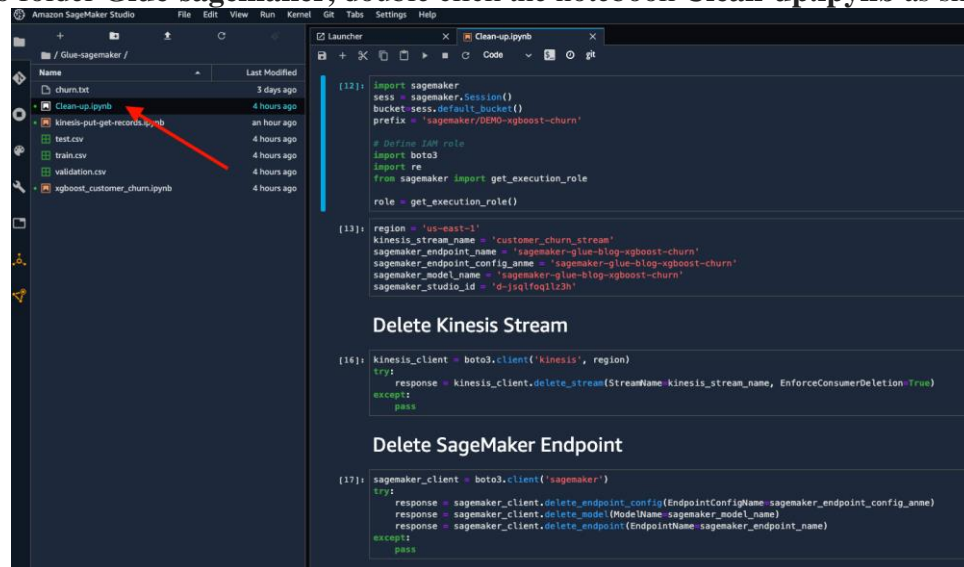
Cleaning up

To avoid incurring future charges, delete the resources used in this post after you are done with steps above.

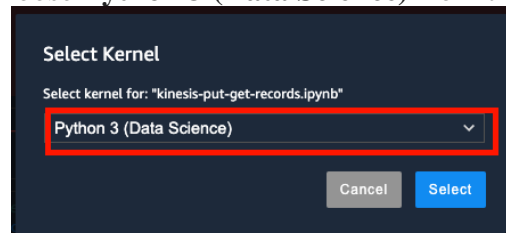
Delete Kinesis Data Stream and SageMaker Endpoint

You can delete the Kinesis data stream and SageMaker endpoint by running the notebook **Clean-up.ipynb**.

1. Go to SageMaker Studio, **Open Studio** for the user **default-glue-sagemaker**.
2. Go to the repo folder **Glue-sagemaker**, double click the notebook **Clean-up.ipynb** as shown below.



3. For **Select Kernel**, please choose **Python 3 (Data Science)** from the dropdown list.



4. Run all cells. If there are no error, the Kinesis data stream and SageMaker endpoint have been deleted.

Delete SageMaker Studio Domain (Studio)

For deleting the SageMaker Studio domain, the on-line SageMaker documentation to delete all resources on the SageMaker Studio domain can be found [here](#).

To delete a domain

1. Open the [SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page to open the **Amazon SageMaker Studio Control Panel**.
3. Repeat the steps below for the all users in the **User Name** list:
 - a. Choose the user.
 - b. On the **User Details** page, for each non-failed app in the **Apps** list, choose **Delete app**.
 - c. On the **Delete app** dialog, choose **Yes, delete app**, type *delete* in the confirmation field, and then choose **Delete**.

- d. When the **Status** for all apps show as **Deleted**, choose **Delete user**.

Important

When a user is deleted, you lose access to the Amazon EFS volume that contains data, including notebooks and other artifacts.

4. When all users are deleted, choose **Delete Studio**.
5. On the **Delete Studio** dialog, choose **Yes, delete Studio**, type *delete* in the confirmation field, and then choose **Delete**.

Conclusion

In this post, you saw how to prepare your own ML endpoint running on SageMaker and create Glue batch and real-time streaming jobs to invoke the ML endpoint to add prediction results along with the original input data set. In this way, you enable a capability to build an analytics pipeline with a power of the machine learning (ML) by leveraging AWS Glue jobs and SageMaker ML endpoints.

Glue is a serverless data integration service on AWS. It is powered by Spark distributed computing framework, which can be also selected in Amazon EMR service. Thus, creating Amazon EMR processes to invoke SageMaker ML endpoints can be similar to the work demonstrated in this post.